Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)

Extrapolating Paths with Graph Neural Networks

Jean-Baptiste Cordonnier and **Andreas Loukas**
École Polytechnique Fédérale de Lausanne
{jean-baptiste.cordonnier, andreas.loukas}@epfl.ch

Abstract

We consider the problem of path inference: given a path prefix, i.e., a partially observed sequence of nodes in a graph, we want to predict which nodes are in the missing suffix. We focus on natural paths occurring as a by-product of the interaction of an agent with a network—a driver on the transportation network, an information seeker in Wikipedia, or a client in an online shop. Our interest is sparked by the realization that, in contrast to shortest-path problems, natural paths are usually not optimal in any graph-theoretic sense, but might still follow predictable patterns. Our main contribution is a graph neural network called GRETEL. Conditioned on a path prefix, this network can efficiently extrapolate path suffixes, evaluate path likelihood, and sample from the future path distribution. Our experiments with GPS traces on a road network and user-navigation on Wikipedia confirm that GRETEL can adapt to graphs with very different properties, while comparing favorably to previous solutions.

1 Introduction

Can a graph neural network learn to extrapolate paths from examples? Rather than attempting to connect nodes based on some graph-theoretic objective function (e.g., by looking for a shortest path), this work focuses on naturally occurring paths. Such paths appear whenever an agent tries to reach a target by moving between adjacent nodes in a graph. The agent for example may be a driver that is navigating through a road network or a knowledge seeker browsing Wikipedia articles. Given the graph and a partial knowledge of the path our goal is to predict the future position of the agent.

Path inference problems are demanding because natural paths tend to differ qualitatively from shortest paths. The choice of the agent at every step may depend on a number of factors, such as the structural properties of the graph and the conceptual similarity of nodes as perceived by the agent. For instance, when looking for information in Wikipedia it has been observed that seekers' decisions are correlated with their perception of article similarity and degree [West *et al.*, 2009]. At the same time, some form of directionality is involved in path formation, in the sense that an agent's actions can be conditioned on the entire history of its trajectory. Making a parallel to Euclidean domains, a path can be thought as 'straight' when the agent moves towards nodes that are far from where it was in the past and 'circular' when it returns to its starting position. Contrasting our geometric intuition however, the space here is non-Euclidean and what is far or close should be determined in light of the graph structure.

From a deep learning perspective, the main challenge we face is reconciling graph-based approaches with sequential data. Graph convolutional networks (GCN) have exhibited a measure of success at predicting properties of nodes (e.g., the category of a Wikipedia article or the average traffic flow in a road network) [Bruna *et al.*, 2014; Defferrard *et al.*, 2016; Kipf and Welling, 2016] or of the graph as a whole (e.g., the solubility of a molecule or the functional similarity of two proteins) [Hamilton *et al.*, 2017; Xu *et al.*, 2018]. The isotropic nature of graph convolution however renders it a poor fit for sequential data. On the other hand, sequence prediction problems are typically solved with Recurrent Neural Networks (RNN) [Hochreiter and Schmidhuber, 1997]. These are ideal for pure sequences, such as sentences or time-series, but do not take into account the graph structure.

With this in mind, the main contribution of this paper is GRETEL, a graph neural network that acts as a generative model for paths. We teach our network to modify a graph so that it encodes the directionality of an observed path prefix. Candidate suffixes are then generated by a non-backtracking walk on the modified graph. GRETEL's simple form comes with a number of benefits. Inference can be done efficiently and in closed-form. In addition, the network can be trained to estimate the true path likelihood from very little data.

We evaluate the validity of our approach in two diverse tasks: extrapolating GPS traces on a road network and predicting the Wikipedia article a player is targeting in the game of Wikispeedia [West *et al.*, 2009]. As confirmed by our experiments, GRETEL is highly anisotropic in its operation, despite being based on graph convolution. It also compares favorably to state-of-the-art RNN and other baselines that do not fully exploit the graph structure. GRETEL identifies the correct path $\sim$28% more frequently than the best RNN in the GPS trace experiment and achieves an 8-fold target accuracy improvement in the Wikispeedia dataset at 3 hops.

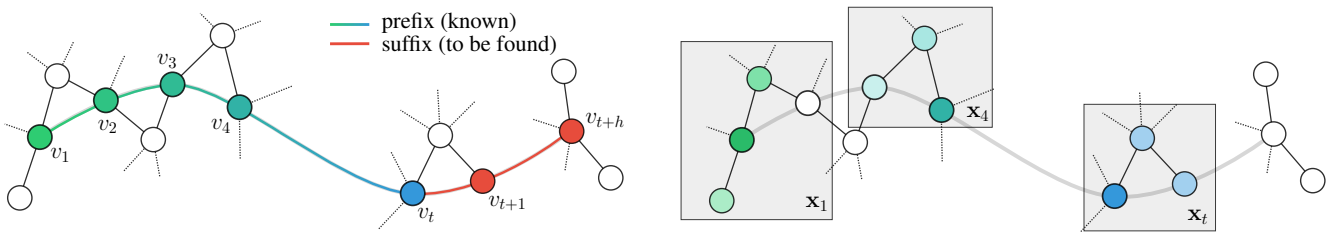


Figure 1: The path inference problem entails predicting a path suffix $s = (v_{t+1}, \ldots, v_{t+h})$ given a graph and a prefix $p = (v_1, \ldots, v_t)$ (left). In the generalized formulation, a trajectory ϕ encodes the approximate position of a subset of nodes in the prefix (right).

2 The Path Inference Problem

Suppose that there exists an agent[1] that navigates a directed graph $G = (\mathcal{V}, \mathcal{E})$ consisting of $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. The agent occupies a single vertex at a time t and it may take one step to move between node v_i to v_j whenever a directed edge $e_{i \to j} \in \mathcal{E}$ exists. Its position across time is summarized by the traversed path $(v_1, v_2, \ldots, v_t)$, which is a sequence of nodes that are pairwise adjacent (see Figure 1, left). We write $\vec{e}_t$ and $\overleftarrow{e}_t$ to refer respectively to the edges in the forward and backward path order: forward edge $\vec{e}_t$ goes from $v_t \to v_{t+1}$, whereas $\overleftarrow{e}_t$ traverses the path in the opposite direction, from $v_t \to v_{t-1}$.

Path inference. The problem we consider entails estimating the likelihood $\mathrm{P}(s \,|\, p, G)$ of a *suffix* path $s = (v_{t+1}, \ldots, v_{t+h})$ given a *prefix* path $p = (v_1, \ldots, v_t)$ on graph G. The likelihood may additionally depend on an assortment of available features relating to nodes, edges, and the agent itself. Further, since the number of possible paths increases exponentially with the prediction horizon h, it is also important to find efficient ways of (*i*) sampling paths, and (*ii*) identifying the one that has the maximum likelihood. In certain situations, one may also be interested in (*iii*) the marginal likelihood of v_{t+h}, i.e., the probability that the agent reaches v_{t+h} after h steps.

Generalization. The above formulation assumes that the path prefix is exactly known—a requirement that may not be met in practice. To this end, we generalize the path inference problem in two ways. First, we suppose that we possess only an approximate idea of the agent's position. For every t, we represent our knowledge by a vector $\mathbf{x}_t \in \mathbb{R}^n_{\geq 0}$, conveniently normalized to have measure one. The i-th entry of $\mathbf{x}_t$ is then interpreted as the likelihood that the agent resides at node v_i at step t. This comes handy also if we try to extrapolate a path recursively, as subsequent calls take into account the uncertainty of previous decisions. Second, we posit that only a subset of the agent's path, called a trajectory, can be observed. Let $\mathcal{I}$ be a sub-sequence of $(1, 2, \ldots, t)$. A *trajectory*

$$\phi \stackrel{\text{def}}{=} (\mathbf{x}_\tau : \tau \in \mathcal{I})$$

is then a sub-sequence of $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t)$. Since ϕ is always defined in terms of $\mathcal{I}$, whenever a function has access to ϕ in the following we assume that it also knows $\mathcal{I}$ (though this remains implicit in the notation). With this in place, the *generalized path inference* problem amounts to estimating the likelihood $\mathrm{P}(s \,|\, \phi, G)$ of a path suffix $s = (v_{t+1}, v_{t+2}, \ldots, v_{t+h})$ given a trajectory ϕ and the graph G.

3 Finding Paths with GRETEL

We wish to construct a generative model for paths. Given an input trajectory and a horizon the model should be able to generate candidate suffixes and inform us of their likelihood.

A key challenge in this pursuit lies in capturing directionality. Setting aside the special case of product graphs[2], most graphs are not imbued with a natural notion of direction. This is also the reason why graph convolution is an isotropic operation: from a graph-theoretic perspective, there is no consistent way of ordering or grouping the neighbors of any given node. Fortunately, what we refer to in jest as "curse of directionality" can be broken if one combines the graph structure with additional information, such as a path prefix: given a path of length t, every node becomes capable of separating its neighbors in up to[3] $3t$ groups depending on whether they are closer, equidistant, or further from each node in the path.

Armed with this intuition, our approach will be to train a graph neural network f_θ called GRETEL to encode all available information about a path prefix into a *latent graph*

$$\Phi \stackrel{\text{def}}{=} (\mathcal{V}, \mathcal{E}, w_\phi), \quad \text{with} \quad w_\phi = f_\theta(G, \phi).$$

Though Φ shares the same vertex and edge sets as G, its edges are re-weighted so as to point towards the directions the agent is most likely to follow. We will then approximate the likelihood of any suffix s by the graph-dependent model g, as follows:

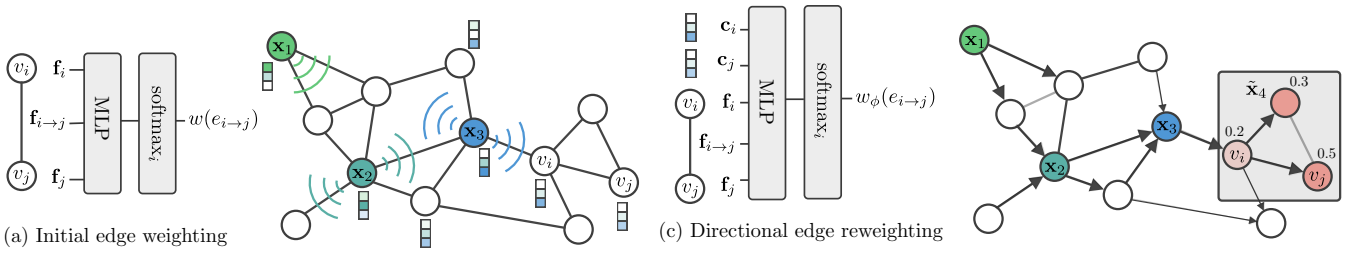
$$\mathrm{P}(s \,|\, h, \phi, G) \simeq g(s, \mathbf{x}_t, \Phi),$$

where $\mathbf{x}_t$ captures the last known position of an agent. The notation above indicates that the model sees Φ instead of G. We define the encoder f_θ and generator g in the following sections and represent the end-to-end architecture in Figure 2.

[1] Though in some cases an actual agent might not exist, the path inference problem becomes more intuitive if we pretend that it does.

[2] A product graph can be expressed as the graph product of k simpler graphs. For such graphs, one may define k consistent notions of direction, each corresponding to one constituent graph. Examples include the grid and hyper-cube (Cartesian product), Rook's graph (Tensor product), and King's graph (Strong product).

[3] The exact number depends on the level-sets of the distance function for every node in the path.



(a) Initial edge weighting

(b) Encoder f_θ diffuses each observation $\mathbf{x} \in \phi$ to compute each node's pseudo-coordinates.

(c) Directional edge reweighting

(d) Generator g starts a non-backtracking RW from $\mathbf{x}_3$ on latent directional graph Φ.

Figure 2: Extrapolation of a trajectory $\phi = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ (colored from green to blue) on the original graph G. Parametrized encoder f_θ computes the re-weighted latent directional graph $\Phi = (G, w_\phi)$ (a-c) and generator g computes next step distribution $\tilde{\mathbf{x}}_4$, shown in red (d).

3.1 Capturing Direction

We train a graph neural network $f_\theta(G, \phi)$ to predict the most likely direction in the vicinity of every node. The network assigns a weight to every directed edge by combining available features with a system of learned pseudo-coordinates (equivalently embeddings), capturing the relation between every node and observation in the prefix.

The pseudo-coordinate vectors $\mathbf{c}_i = [\mathbf{c}_{i,1}, \dots, \mathbf{c}_{i,|\phi|}]$ characterizing each node v_i are jointly learned by a graph convolutional network (GCN) of k-layers:

$$\mathbf{c}_{i,\tau} = [\text{GCN}(\mathbf{x}_\tau)]_{i,:} \quad \text{for} \quad \tau \in \mathcal{I}.$$

The GCN computes the τ-th pseudo-coordinate vector in parallel for all nodes in a recursive manner:

$$[\mathbf{X}_k]_{i,:} = \text{ReLU}\left(\sum_{v_j \in \mathcal{V}} w(e_{j\to i}) \, [\mathbf{X}_{k-1}]_{j,:} \mathbf{W}_k \right),$$

with $\mathbf{X}_0 = \mathbf{x}_\tau$. In practice we noticed that using a univariate non parametrized diffusion gave good result and we set $\mathbf{W}_k$ to 1. Each edge weight is initialized by a multi-layer perceptron $w(e_{i\to j}) = \text{MLP}(\mathbf{f}_i, \mathbf{f}_j, \mathbf{f}_{i\to j})$ taking as input the features of nodes v_i and v_j as well as those corresponding to edge $e_{i\to j}$. Finally, the weight $w_\phi(e_{i\to j})$ of every directed edge is decided by a simple network that predicts the most likely direction. We use once more a multi-layer perceptron

$$z_{i\to j} = \text{MLP}(\mathbf{c}_i, \mathbf{c}_j, \mathbf{f}_i, \mathbf{f}_j, \mathbf{f}_{i\to j}),$$

followed by a soft-max over outgoing edges

$$w_\phi(e_{i\to j}) = \frac{z_{i\to j}}{\sum_{v_l \in \mathcal{V}} z_{i\to l}}.$$

The latter ensures that the weights of all out-going edges of each v_i can be interpreted as a categorical distribution. Akin to skip connections in residual networks, we reuse features in both MLP so as to facilitate training.

3.2 A Generative Model with Short-Term Memory

We opt for a generative model that performs a *non-backtracking walk* on the latent graph. Akin to a random walk, the model assumes that the agent traverses each forward edge $\vec{e}_\tau$ from v_τ to $v_{\tau+1}$ with probability proportional to the learned edge weight $w_\phi(\vec{e}_\tau)$. The main difference is that the walk cannot return to its previous position (i.e., $\vec{e}_\tau \neq \overleftarrow{e}_\tau$).

Using a model with short-term memory has two interesting consequences. First, the graph neural network is encouraged to find a meaningful latent graph, capturing the directionality of the path. At the same time, inference can be done in closed-form (and often efficiently), greatly simplifying training. We provide three examples in the following:

Suffix likelihood. The likelihood of a path suffix $s = (v_{t+1}, \dots, v_{t+h})$ is

$$g(s, \mathbf{x}_t, \Phi) = \sum_{v_t \in \mathcal{V}} \left(\prod_{\tau=t}^{t+h-1} p_\tau \right) [\mathbf{x}_t]_{v_t},$$

where $\mathbf{x}_t$ captures the last known position of an agent and the non-backtracking probabilities p_τ are

$$p_\tau \overset{\text{def}}{=} \begin{cases} w_\phi(\vec{e}_\tau) & \text{if } \tau = t, \\ \dfrac{w_\phi(\vec{e}_\tau)}{1 - w_\phi(\overleftarrow{e}_\tau)} & \text{if } \vec{e}_\tau \neq \overleftarrow{e}_\tau \text{ and } \tau > t, \\ 0 & \text{o.w.} \end{cases}$$

It can be seen that the computational complexity grows linearly with the support of $\mathbf{x}_t$ and the horizon h. Hence, when aiming to extrapolate paths we can efficiently train our network by minimizing directly the negative log-likelihood (NLL) of the true suffix.

Target likelihood. Alternatively, we can train our network to predict the distribution $\mathbf{x}_{t+h}$ of the target over a known horizon. Following [Kempton, 2016], let $\mathbf{P}_\phi$ be the $m \times m$ non-backtracking matrix with

$$[\mathbf{P}_\phi]_{e_{i\to j}, e_{k\to l}} = \begin{cases} 0 & \text{if } j \neq k \text{ or } i = l, \\ \dfrac{w_\phi(e_{k\to l})}{1 - w_\phi(e_{k\to i})} & \text{o.w.} \end{cases}$$

Further, define the $m \times n$ matrix $\mathbf{B}_\phi$, with $[\mathbf{B}_\phi]_{e_{i\to j}, k} = 0$ if $k \neq i$ and $w_\phi(e_{k\to j})$ otherwise. The marginal distribution $\hat{\mathbf{x}}_{t+h}$ of the non-backtracking walk on Φ after h steps can be written as

$$\hat{\mathbf{x}}_{t+h} = \mathbf{B}_\phi^+ \, \mathbf{P}_\phi^h \, \mathbf{B}_\phi \, \mathbf{x}_t,$$

where due the special sparsity structure of $\mathbf{B}_\phi$ (its rows have disjoint support), the pseudo-inverse $\mathbf{B}_\phi^+$ is, up to normalization, equal to $\mathbf{B}_\phi^\top$. The computational complexity is thus linear $O(mh)$ w.r.t. the number of edges and horizon. The network can be trained by minimizing the cross entropy between $\hat{\mathbf{x}}_{t+h}$ and $\mathbf{x}_{t+h}$ or any other measure between distributions.

Most likely suffix. The maximum likelihood suffix over a given horizon can be identified by Monte-Carlo sampling or deterministically. Suppose that the agent resides at node v_t almost surely. Further, let H_ϕ be the weighted directed graph whose adjacency matrix is $\log(\mathbf{P}_\phi)$ (the logarithm is applied only to non-zero entries of $\mathbf{P}_\phi$). Since the nodes of H_ϕ correspond to edges in G, a suffix s can also be seen as a path $(\vec{e}_t, \ldots, \vec{e}_{t+h-1})$ on H_ϕ. Moreover, as a consequence of the transformation, the weight of s in H_ϕ is the same as its log-likelihood:

$$\log(g(s, \boldsymbol{\delta}_t, \Phi)) = \log\left(w_\phi(\vec{e}_t) \prod_{\tau=t}^{t+h-2} [\mathbf{P}_\phi]_{\vec{e}_\tau, \vec{e}_{\tau+1}}\right)$$
$$= \log(w_\phi(\vec{e}_t)) + \sum_{\tau=t}^{t+h-2} \log\left([\mathbf{P}_\phi]_{\vec{e}_\tau, \vec{e}_{\tau+1}}\right),$$

where $\boldsymbol{\delta}_t$ is a dirac centered at v_t. The most likely suffix is therefore identified in a deterministic manner by performing a best-first traversal starting from v_t and searching for the maximum-weight path of length h. The computational complexity is $O(\Delta^h)$, where Δ is a bound on the maximum degree in the h-hop neighborhood of v_t (this bound is tight for a perfect Δ-ary tree of depth h with all edge weights being equal).

4 Experiments

Our goal is two-fold. First, in Section 4.1 we wish to confirm that GRETEL can capture the directionality of (straight) paths in the edges of the latent graph. In addition, we are interested in evaluating the generality of our approach and its performance on real data. This is pursued by taking on two diverse tasks: GPS trace extrapolation in Section 4.2 and user-navigation on a knowledge network in Section 4.3. Information about the datasets and hyper-parameters are displayed in Table 1, code and datasets are publicly available at https://github.com/jbcdnr/gretel-path-extrapolation.

4.1 Can GRETEL Learn a Straight Path?

We constructed a toy experiment to qualitatively test whether GRETEL has the capacity to capture directionality in the Euclidean sense. Specifically, we generated straight trajectories on a random graph built to approximate a plane (by uniformly sampling $n = 500$ points in $[0, 1]^2$ and applying a 10-NN construction). The trajectories were obtained by mapping straight lines to the closest nodes and sub-sampling the resulting path.

Four typical runs of the experiment are shown in Figure 3. Given a trajectory (disks from blue to green), GRETEL was trained to predict the target (green circle) by minimizing a target cross entropy loss. The task is non trivial as GRETEL is not given the positions of the nodes. In addition, the graph differs from a regular grid and does not offer a good approximation of the underlying Euclidean space.

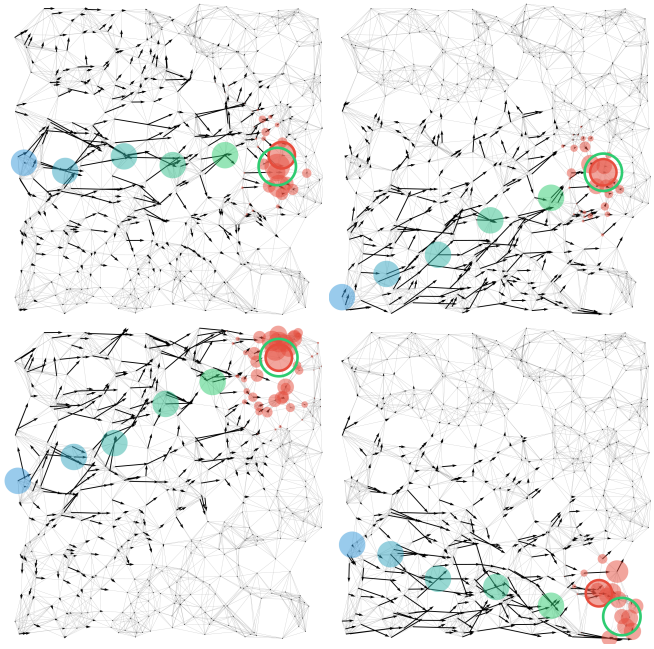


Figure 3: Extrapolation of a straight trajectory. The input trajectory is visualized by disks whose color varies from blue to green. True target is highlighted by a green circle. GRETEL's predicted target distribution is shown with red disks and the maximum likelihood target with a red circle. Arrows indicate chosen edges at each node, length represents confidence.

As seen in Figure 3, most of the probability mass (red disks) of the predicted distribution $\hat{\mathbf{x}}_{t+h}$ is concentrated close to the target. Moreover, as intended, the direction of the trajectory is encoded into the edge weights of Φ, despite the sampling irregularity (note that a black arrow indicates the most significant edge at each node). In the bottom right figure, due to the existence of a hole between the end of the trajectory and the target, the graph neural network assigns small likelihood to the correct target. We hypothesize that the phenomenon is exaggerated by the graph being dense near the north hole boundary, which causes the learned distance metric (implied by the learned pseudo-coordinates) to locally deviate from the Euclidean distance. It is also intriguing to observe that, whereas in the leftmost figure GRETEL does not identify correctly the target, the neural network's answer presents a visually plausible alternative.

4.2 GPS Trace Extrapolation

In the GPS trace extrapolation problem, we observe a prefix of ordered GPS locations emitted by a driver moving on the road network. Two distinct objectives can be addressed: (*i*) predict the position of the driver in h seconds, or (*ii*) predict the following roads that the driver will follow.

Figure 4 illustrates visually the two scenarios. Two trajectories are shown (blue to green filled circles) along with the

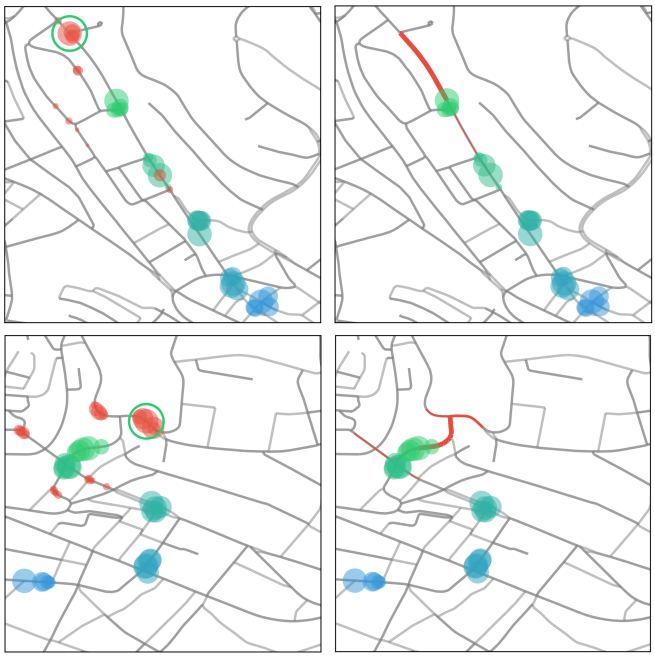


Figure 4: Two examples of GPS trace extrapolation. Past trajectories are composed of 5 observations (blue to green). *Left:* target distribution as red disks, green circle is the true target. *Right:* sampled future suffixes in red.

output of GRETEL for each objective (red): the target distribution is on the left and the sampled likely suffixes are on the right. Larger markers/bolder lines correspond to more likely targets according to our model. The goal is to predict the true target (highlighted by a green circle) and extrapolate the trajectory towards it.

Existing solutions. The classical approach to learn patterns from navigation-traces is to model them by a Markov Decision Process and learn the transition probabilities from the observed data. The Markovian property can also be relaxed by taking into account multiple steps at a time (akin to n-grams). The main issue with such approaches is their sample complexity—accurately estimating the probability of rare state transitions requires prohibitive amount of data as the number of parameters grows in the best case (i.e., even when $n = 1$) linearly with the number of edges. More recently, the GPS extrapolation problem was solved by a recurrent neural network (RNN), achieving state-of-the-art performance [Wu *et al.*, 2017]. The architecture in question resembles a standard RNN with the main difference that it integrates the choice restrictions induced by the road network at each step.

Experimental setup. We ran an experiment based on a small dataset of food deliveries (229 traces) occurring over the OpenStreetMap[4] road network of Lausanne (18156 nodes, 32468 edges). We mapped the GPS coordinates to the $k = 5$ closest intersection nodes. The trajectories were preprocessed to have consecutive observations at least 50 meters apart. The min/max/median node degree was 1/6/2. Edge features were speed limit and length. It is important to note

[4]https://openstreetmap.org

	GPS	Wikispeedia
# nodes	18'171	4'605
# edges	32'491	119'883
# trajectories	229	28'011
train/test	80% / 20%	80% / 20%
prefix (# observations)	5	4
horizon (# nodes)	9.1 ± 3.6	1, 2, 3
first edge MLP	none	6 - 12 - 1
second edge MLP	12 - 24 - 1	14 - 28 - 1
activation type	sigmoid	sigmoid
# GCN layers (k)	60	5
Adam learning rate	0.01	0.1
batch size	5	10
# epochs	200	5

Table 1: Dataset information, model and training hyper-parameters. Code and datasets are publicly available at https://github.com/jbcdnr/gretel-path-extrapolation.

Model	NB	Loss	Choice acc ↑	$P(v_{t+h})$ ↑	NLL ↓
Uniform			31.5	0.035	2.40
Uniform	✓		48.8	0.133	1.89
GRETEL		target	63.6 ± 2.8	0.110 ± 0.006	1.94 ± 0.08
		NLL	65.6 ± 0.8	0.108 ± 0.008	1.87 ± 0.05
GRETEL	✓	target	74.2 ± 1.4	$\mathbf{0.199} \pm 0.003$	$\mathbf{1.28} \pm 0.04$
	✓	NLL	68.8 ± 2.2	$\mathbf{0.199} \pm 0.004$	1.50 ± 0.04
CSSRNN* 50		NLL	73.9 ± 1.6		1.67 ± 0.08
CSSRNN* 4			66.2 ± 2.6		1.57 ± 0.01
LPIRNN* 50			74.2 ± 3.1		1.53 ± 0.06
LPIRNN* 4			$\mathbf{75.0} \pm 2.3$		1.87 ± 0.02

Table 2: Results of GPS trace extrapolation on test dataset. Choice accuracy (%) is computed at non trivial intersections only (more than two outgoing roads). We present an ablation of the non-backtracking (NB) property of GRETEL's random walk. Target probability is not given for the RNN. We use an asterisk to indicate which algorithms have access to the road coordinates.

that a GPS trace corresponds to a sequence of noisy physical locations and not a sequence of adjacent nodes. Nevertheless, all methods discussed so far require a path in order to function properly. Hence, as an extra pre-processing step, for all baselines the GPS traces were mapped to paths using a Hidden Markov Model (HMM) [Newson and Krumm, 2009]. To test the versatility of our approach, we did not employ the map-matching algorithm with GRETEL, but provided it with the raw trajectories as input. We used a non-parametric diffusion in the encoder as a learned GCN did not improve the performance. We trained the RNN based models from [Wu *et al.*, 2017] on one jump extrapolation objective instead of the full suffix so that it is exposed to the same samples as our method.

Table 2 reports extrapolation accuracy w.r.t. three measures. First, *choice accuracy* measures how accurate are the decisions of an algorithm at each crossroad of the ground-truth path connecting v_t and v_{t+h}, as extracted by the HMM. We computed the choice accuracy on only nodes whose degree was at least 3, as the decision is trivial otherwise. As seen by the accuracy of a uniform and non-backtracking random-walk, most crossroads encountered had degree 3, leading to a random decision succeeding ∼31.5% of the time.

RNN models [Wu *et al.*, 2017] reached a good choice accuracy on the test set, slightly outperforming GRETEL (the difference is smaller than the standard-deviation across 5 independent runs). However, they were less competitive in recovering the actual suffix, as measured by the *negative log-likelihood* (NLL) measure. We note that choice accuracy is more lenient with sporadic mistakes as compared to the suffix NLL: the likelihood of a path depends heavily on the worst decision made, whereas this is not true for the former. To confirm that the RNN were not affected by overfitting, we repeated the experiment with a smaller hidden size representation of 4 (instead of 50 as proposed by the authors). This brought about only a small improvement to the CSSRNN architecture with the NLL dropping from 1.67 to 1.57 and did not help the LPIRNN.

Despite achieving moderate choice accuracy, GRETEL was able to guess the correct suffix $\sim$28% more frequently than the best RNN: in terms of geometric mean, the two algorithms assigned 0.278 and 0.216 probability to the correct suffix, respectively. This is surprising as the RNNs were given a competitive advantage by knowing the road coordinates, whereas GRETEL did not. Interestingly, the best result was attained when the graph neural network was trained to locate the target, even when measuring NLL on the test set. Our hypothesis is that choosing a target loss encourages the neural network to explore alternative suffixes towards the target early on, thus improving training.

Finally, we report the *target probability* measure $\mathrm{P}(v_{t+h})$ corresponding to the average chance an algorithm has to find a node v_i with non-zero $[\mathbf{x}_{t+h}]_i$ (due to the GPS-to-node mapping procedure, there were five such nodes for each trajectory). We were unable to incorporate this metric for the RNN, as the implementation provided by the authors does not support auto-regressive sampling. In our test set, GRETEL was able to find the target $\sim$20% of the time, outperforming simple baselines.

4.3 User Navigation in Wikipedia

In the Wikispeedia game [West *et al.*, 2009], human players are called to find a path from a source to a target article by following a sequence of hyperlinks. Since players can only view the available links locally and guess links on other pages based on their prior knowledge, most paths taken by players differ qualitatively from shortest-paths on the graph of articles [West and Leskovec, 2012]. Therefore, it is intriguing to determine whether an algorithm can learn to mimic the human routing logic. *In particular, given a path prefix can we predict towards which article the player is navigating towards—or perhaps human choices are too unpredictable?*

Motivated by this question, we trained GRETEL to predict the target of a navigation prefix among Wikipedia articles. We optimized both objectives, the target probability cross entropy and the suffix negative log-likelihood. Node features were the node in/out degrees (to capture the notion of hubs), while edge feature vectors contained the TF-IDF similarity between source and destination articles of each hyperlink along with the number of times this link was clicked in the training dataset of paths. We compare to previous work and baselines which only use article features (e.g. TF-IDF vectors) and local edge features (e.g. node degrees and properties of edges in the prefix). On the contrary, our method sees the entire graph (not only the local connections of nodes in the suffix), which we will argue is essential to solving this problem.

	precision@1			2-targets accuracy		
Path length n	5	6	7	5	6	7
Uniform NB	1.9	0.1	0.0	49.6	67.3	58.2
Reweighted	15.9	3.8	0.6	77.1	84.2	79.8
FastText	3.0 $\pm$ 0.1	0.7	0.2	68.9 $\pm$ 0.5	70.8 $\pm$ 0.5	68.2 $\pm$ 0.6
West, 2012				80	84	80
GRETEL target	**19.5** $\pm$ 1.3	**6.2** $\pm$ 0.3	**4.9** $\pm$ 0.3	**82.2** $\pm$ 0.3	**89.2** $\pm$ 0.3	**84.6** $\pm$ 0.3
GRETEL NLL	19.0 $\pm$ 1.4	6.1 $\pm$ 0.4	4.0 $\pm$ 1.1	81.6 $\pm$ 0.3	88.5 $\pm$ 0.2	83.9 $\pm$ 2.9

Table 3: Wikispeedia target prediction accuracy (%) given path prefix of 4 articles. Precision@1 is the ratio of true targets that are assigned the highest probability. 2-targets accuracy is computed on the classification between the true target and a random article at the same distance. The length of the suffix is $n - 4$. Standard-deviation has been computed on the basis of 5 independent runs.

Previous work. West and Leskovec [West and Leskovec, 2012] consider two variants of the target prediction task problem: (*i*) given a path prefix, the target and a negative target sampled randomly, predict which one is the true target; and (*ii*) given a prefix, rank all the possible targets. The code is not publicly available, but we report the accuracy of their 2-targets classifier. They extracted carefully tuned features to mimic user way finding, considering for example node degrees for hubs and semantic similarity improvement over the path. Their precision metric to evaluate the ranking model considered sibling articles (same sub-category as the target) as correct predictions, whereas we were less lenient in our evaluation and only considered the prediction correct if the true target was found.

Other baselines. We trained a simple predictor based on FastText pre-trained word embeddings of dimension 300 [Mikolov *et al.*, 2018]. Article feature vectors $\mathbf{f}_i$'s were the average of their word representations. Given a prefix $(v_1, \ldots, v_4)$, the model computed for each target $\hat{y}_j = \sum_{i=1}^{4} \mathbf{f}_i^\top \mathbf{W}_i \mathbf{f}_j$, followed by a soft-max to represent a categorical probability over the nodes. The parameters (i.e., $\mathbf{W}_i$) were trained using Adam, with the learning rate set to 0.01. This baseline shows what can be achieved without any knowledge of the graph and using only article semantic similarities. For instance, according to FastText, the closest articles to "Moon" are "Mercury", "Venus", "Earth's atmosphere", "Shackleton crater" and "Mars". Finally, we compare to two non-parametric versions of GRETEL: (Uniform NB) a non backtracking random walk run for the distance of the path on the random walk graph starting from v_t and (Reweighted) a random walk that has been positively biased towards following frequent links, i.e., those that players favored in the training set.

Quantitative results. Table 3 reports two metrics: *precision@1* measures how often a classifier recovers the actual target, whereas *2-targets accuracy* tests if a classifier can dis-

Prefix, **true suffix**, *sampled suffices from* GRETEL	$\mathrm{P}(s \mid p, G)$
Lunar eclipse, Sunlight, Electromagnetic radiation, Atom	
Nuclear fission, Nuclear power	
Chemical element, Periodic table	0.13
Nuclear fission, Nuclear power	0.10
Nuclear fission, Nuclear weapon	0.07
Latvia, Russia, People's Republic of China, Nepal	
Himalayas, Edmund Hillary	
Mount Everest, Tenzing Norgay	0.20
Mount Everest, Edmund Hillary	0.12
Himalayas, Yeti	0.06
Lawrencium, Russia, United States, Publishing	
Newspaper, The Wall Street Journal	
Book, Library	0.20
Book, Novel	0.11
Newspaper, The Wall Street Journal	0.07

Table 4: Handpicked examples of the top-3 most likely suffixes according to GRETEL in the Wikispeedia test set.

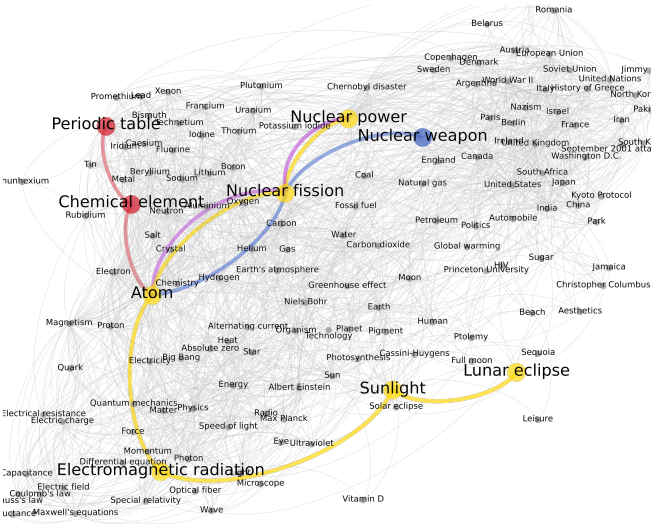


Figure 5: Visualization of a path (yellow) on a subset of the Wikipedia graph along with the top three predictions of GRETEL (red, purple, blue in decreasing likelihood). Aiming to improve visibility, we display only the one-hop neighbors of the nodes in the true path. The graph layout selected was the Force-Layout 2 implemented in the gephi software [Bastian *et al.*, 2009]. Small perturbations were introduced to node positions to minimize label occlusion.

tinguish the true target from a random article selected from those in the same shortest path distance as the true target and given at least once as target (mimicking [West and Leskovec, 2012]). As seen, GRETEL achieves between 4%-and-6% absolute accuracy improvement over state-of-the-art. We attribute this improvement to it considering each path prefix in light of the full graph between articles: nodes close to the prefix path can be discarded as possible targets as the player would probably have found them or stayed in their close neighborhood. This notion of proximity is not accessible to other methods, while we believe it is crucial in attaining good accuracy. The FastText method does incorporate some knowledge of the world not accessible to other methods. However, our experiment suggests that the intrinsic meaning of articles does not suffice to make a good prediction. On the other hand, re-weighting the edges based on how frequently they have been used in the training set is a very effective strategy when trying to predict the next article, but suffers for larger horizons. A case in point, if GRETEL was used to suggest to users the next article to look at, it would match their choice 1/5.26 of the time for one hop prediction and 1/20.4 of the time for three hops, whereas the Reweighted baseline would be correct 1/6.28 and 1/166.6 of the time, respectively: for a horizon of three hops, therefore, our method improves target accuracy by $8.16\times$.

Qualitative results. We also report five hand-picked examples from our test set in Table 4 and visualize one of them in Figure 5.

5 Related Work

To the extend of our knowledge, this is the first time the generalized path inference problem has been considered. An interesting relevant work proposed to classify nodes belonging to the shortest path between a source and a target [Battaglia *et al.*, 2018], but this is a combinatorial problem optimizing a well known graph metric, rather than naturally occurring agents' paths.

Our refinement of the graph into a latent graph is inspired from their message passing framework. Other specialized graph convolutional network layers, such as Graph Attention Network [Velickovic *et al.*, 2017], could be also used to tune the edge weights and allow for anisotropic filtering. The main difference from these approaches is that we use a non-backtracking walk as a generative model in order to extrapolate paths.

Random walks on graphs have been used previously in a deep learning context in order to sample paths from graphs and extract node representations [Grover and Leskovec, 2016; Perozzi *et al.*, 2014] using [Mikolov *et al.*, 2013]. We can see the pseudo-coordinates as node representations with regard to the observations, but the similarity stops there.

6 Conclusion

This paper focused on the path inference problem and its generalization to trajectories. We proposed a novel graph neural network architecture combining a GCN and a non-backtracking walk generator. Our model refines a graph to capture directionality by conditioning it on a path prefix. The simplicity of the latent representation allows us to sample suffixes efficiently and compute path and target likelihoods.

The path inference problem has remained relatively unexplored, yet it has many applications among which are GPS trace extrapolation and user navigation in information networks, as shown in this work. We believe that graph neural networks present a promising solution. We are very interested in determining the limits of their ability.

Acknowledgments

We thank the Swiss National Science Foundation (SNSF) for supporting this work (project "*Deep Learning for Graph-Structured Data*", grant number PZ00P2 179981).

References

[Bastian *et al.*, 2009] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. In *International AAAI Conference on Weblogs and Social Media*, 2009.

[Battaglia *et al.*, 2018] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[Bruna *et al.*, 2014] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014)*, April 2014.

[Defferrard *et al.*, 2016] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3837–3845, 2016.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864, New York, NY, USA, 2016. ACM.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[Kempton, 2016] Mark Kempton. Non-backtracking random walks and a weighted ihara's theorem. *Open Journal of Discrete Mathematics*, 6(04):207, 2016.

[Kipf and Welling, 2016] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[Mikolov *et al.*, 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[Mikolov *et al.*, 2018] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[Newson and Krumm, 2009] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. pages 336–343, November 2009.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, 2014.

[Velickovic *et al.*, 2017] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017.

[West and Leskovec, 2012] Robert West and Jure Leskovec. Human wayfinding in information networks. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 619–628, New York, NY, USA, 2012. ACM.

[West *et al.*, 2009] Robert West, Joelle Pineau, and Doina Precup. Wikispeedia: An online game for inferring semantic distances between concepts. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1598–1603, 2009.

[Wu *et al.*, 2017] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. Modeling trajectories with recurrent neural networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3083–3090, 2017.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.